

1 Dynamic Modularity in Flexible, Persistent Agents

2
3 The present invention relates to software agents and in
4 particular to the dynamic deployment of functionality in
5 software agents.

6
7 Agents represent a means of representing a user in the
8 electronic world, bringing together all the various
9 functions that the user wants to perform in that
10 electronic world, including:

- 11 • transient, anonymous presence of an online search;
12 • persistent occasional presence of online shopping at a
13 particular store;
14 • persistent passive presence of directed marking;
15 • persistent though temporary realtime presence in an
16 online game;
17 and many more.

18
19 US Patent Application Publication Number US2002062334 to
20 Hewlett Packard Company, CO, USA, entitled "Dynamic
21 Agents for Dynamic Service Provision" teaches of dynamic
22 agents and a dynamic agent infrastructure that supports
23 dynamic behaviour modification of agents. One key

1 limitation of the approach presented is that it refers to
2 the dynamic deployment of agent behaviour.- However, for
3 the system to work as described, that dynamically
4 deployed behaviour must be predefined (so that individual
5 programs can know what to expect from others). The term
6 'dynamic' is thus misleading. The work could better be
7 described as the dynamic deployment of statically defined
8 behaviour. One problem with US2002062334 is that software
9 engineering in the large, and in particular, multi-party
10 and multi-site development is made much harder by the
11 need to conform to a rigid, pre-specified sets of
12 functionality. It would be advantageous for new
13 functionality to be definable dynamically, so that
14 separate teams can work more independently, thereby
15 simplifying problem decomposition.

16
17 US2002062334 describes dynamic functionality in an agent.
18 It does not decouple functionality and therefore does not
19 support functionality that is not at least defined at
20 agent start-up. It discloses that the dynamic agent's
21 modifiable portion includes its data, knowledge and
22 action programs, which determines its application
23 specific capabilities. Thus the generic, fixed portion of
24 a dynamic agent, together with its application programs,
25 acts as an autonomous problem solver. This means that
26 generic capabilities (such as reasoning) are fixed at the
27 point of startup of an agent. For agents that persist
28 over a long period (months and years, rather than hours
29 or days) this means that it is impossible to introduce
30 new, generic capabilities as they become available.

31
32 Also, US2002062334 does not explain how existing
33 functionality might be replaced. Crucially, if demands

1 are made of a carried action, and those demands pile up,
2 and the carried action is then replaced, there is no way
3 of co-ordinating the rescheduling of those demands for
4 the new carried action. By having a specific component
5 responsible for hand-over during update, it would become
6 possible to queue these demands until the new
7 functionality is in a position to handle them.

8
9 Agents represent individuals, and typically persist over
10 long periods. It is thus a very real problem that adding
11 new generic functionality and upgrading existing
12 functionality in deployed systems should be prohibited.
13 As a consequence, it would be advantageous to implement a
14 method by which even the core, generic functionality can
15 be replaced. A portion of the data of an agent may be
16 retained, at the discretion of the update mechanism, to
17 maintain coherence (so, for example, the beliefs of an
18 agent may persist through an update of core
19 functionality). This would allow upgrading of deployed
20 agents through a straightforward, scalable mechanism.

21
22 US2002062334 discloses that when multiple actions are
23 carried by the same dynamic agent, they can exchange
24 information through the object store of that dynamic
25 agent. However, for a carried action to be able to use
26 data from another module, it must be able to anticipate
27 the availability and internal structure of that data.
28 That is, it must have knowledge of the "interface"
29 provided by that action. So for any two actions that
30 might need to interact, they must (a) know that the other
31 action is carried, (b) know enough about that action to
32 know what data it can provide and in what format. In this
33 way, any 'old' carried action will be unable to exploit

1 the functionality developed in 'new' carried actions that
2 was unanticipated at the time of the development of the
3 'old' carried action. This is a clear theoretical
4 problem that translates into a large practical problem in
5 situations where agent systems are deployed on a wide
6 basis, particularly if they are persistent, and subject
7 to rigorous quality of service requirements (such that
8 they can't simply be 'switched off' for an upgrade).

9
10 A second problem, in a similar vein, is that old actions
11 cannot know of the methods (i.e. published procedures) of
12 a newer carried action. It is presumably for this reason,
13 and because it is much more difficult to design all
14 possible method interfaces on a system-wide basis at the
15 outset, rather than because of its data interfaces, that
16 the disclosed system does not support method calls
17 between carried actions. This reduces the potential for
18 synergistic interplay between carried actions, and can
19 lead to requiring redundancy (reimplementing identical
20 functionality in different carried actions).

21
22 The problem is thus that one carried action cannot make a
23 call upon the abilities of another. This violates key
24 principles of modern programming practice aimed at code
25 reuse and adaptation, by forcing modules to implement all
26 the functionality that they will need, rather than
27 utilising functionality they may know is already
28 available within other modules in the agent.

1 It is an object of the present invention to provide
2 efficient dynamic changes of modules in an agent and
3 overcome at least some of the problems identified above.
4

5 According to a first aspect of the present invention,
6 there is provided an agent for representing a person's
7 identity adapted to dock a dockable module, the dockable
8 module comprising a method means for performing a
9 function, wherein the agent further comprises an
10 intermodule communication means for mapping a request
11 from a requesting module to the method means in the
12 dockable module.
13

14 Preferably said method means is adapted to perform said
15 function responsive to said request.
16

17 Preferably said request from the requesting module
18 comprises a label specifying a function and said method
19 means in said dockable module corresponds to the
20 specified function.
21

22 Preferably said intermodule communication means comprises
23 a store of labels and associated modules.
24

25 Preferably said store of labels and associated modules is
26 a table.
27

28 Optionally said method means is adapted to perform the
29 function of docking said dockable module with said agent.
30

31 Optionally said method means is adapted to perform the
32 function of registering a label with said intermodule

1 communication means, the label specifying a function
2 supported by said dockable module.

3
4 Optionally said method means is adapted to perform the
5 function of undocking said dockable module from an agent.

6
7 Optionally said method means is adapted to perform the
8 function of requesting the discarding of a label from the
9 intermodule communication means, the label specifying a
10 function supported by said dockable module.

11
12 Optionally said method means is adapted to perform the
13 function of updating said dockable module within said
14 agent.

15
16 According to a second aspect of the present invention,
17 there is provided a method of deployment of a module in
18 an agent comprising the steps of:

19 receiving a request for deployment of said module;
20 maintaining registration information relating to a
21 function supported by said module; and
22 invoking said function.

23
24 Preferably said step of maintaining registration
25 information comprises maintaining a store of labels and
26 associated modules, each label specifying a function
27 supported by a module.

28
29 Preferably said store of labels and associated modules is
30 a table.

1 Preferably said step of maintaining registration
2 information comprises the step of registering a label
3 specifying a function supported by said module.
4 This is a final step in the docking process.

5
6 Alternatively said module is a docked module and said
7 step of maintaining registration information comprises
8 the step of unregistering a label specifying a function
9 supported by said docked module.

10 This is a final step in the undocking process.

11 Alternatively said module is a replacement module and
12 said step of maintaining registration information
13 comprises the steps of:

- 14 • unregistering a label specifying a function
15 supported by a docked module from said agent; and
- 16 • registering a label specifying a function
17 supported by said replacement module with said
18 agent

19 and said method further comprises the steps of:

- 20 • storing information related to the state of said
21 docked module; and
- 22 • instantiating said replacement module using said
23 stored information.

24 These constitute steps in the process of updating.

25
26 Optionally, said step of maintaining registration
27 information further comprises the step of queuing calls
28 to the docked modules' registered labels, prior to the
29 step of registering a label specifying a function
30 supported by said replacement module.

31
32 Preferably, said function supported by said module is the
33 method of deployment of said module.

1
2 According to a third aspect of the invention there is
3 provided a computer program comprising program
4 instructions for causing a computer to perform the
5 method of claims 11 to 18.

6
7 In order to provide a better understanding of the present
8 invention, an embodiment will now be described by way of
9 example only and with reference to the accompanying
10 Figures, in which:

11
12 Figure 1 illustrates, in schematic form, an agent and a
13 module in accordance with a preferred embodiment of the
14 present invention;

15
16 Figure 2 illustrates, in schematic form, an overview of
17 agentative representation in a multi-service environment;

18
19 Figure 3 illustrates, in schematic form, the process by
20 which a label is resolved in accordance with a preferred
21 embodiment of the present invention;

22
23 Figure 4 illustrates, in schematic form, module docking
24 in accordance with the present invention;

25
26 Figure 5 illustrates, in schematic form, module updating
27 in accordance with the present invention;

28
29 The inventions are an architecture and methods for
30 dynamic deployment of modules in a software agent.

31
32 Although the embodiments of the invention described with
33 reference to the drawings comprise computer apparatus and

1 processes performed in computer apparatus, the invention
2 also extends to computer programs, particularly computer
3 programs on or in a carrier, adapted for putting the
4 invention into practice. The program may be in the form
5 of source code, object code, a code of intermediate
6 source and object code such as in partially compiled form
7 suitable for use in the implementation of the processes
8 according to the invention. The carrier may be any
9 entity or device capable of carrying the program.

10
11 For example, the carrier may comprise a storage medium,
12 such as ROM, for example a CD ROM or a semiconductor ROM,
13 or a magnetic recording medium, for example, floppy disc
14 or hard disc. Further, the carrier may be a
15 transmissible carrier such as an electrical or optical
16 signal which may be conveyed via electrical or optical
17 cable or by radio or other means.

18
19 When the program is embodied in a signal which may be
20 conveyed directly by a cable or other device or means,
21 the carrier may be constituted by such cable or other
22 device or means.

23
24 Alternatively, the carrier may be an integrated circuit
25 in which the program is embedded, the integrated circuit
26 being adapted for performing, or for use in the
27 performance of, the relevant processes.

28
29 With reference to Figure 1, the architecture 100 of an
30 agent according to the present invention is best
31 visualised as including a torus. On the inside of the
32 torus 102, a special module, the core module 104,
33 attaches itself. On the outside of the torus, any number

1 of application specific modules 106, 108 may also become
2 attached. The security and unity of the agent is also
3 conceptually protected by a thin sphere 110 encompassing
4 all the modules. The torus itself coordinates all
5 communication between modules and between modules and
6 core: this is the Inter Module Communication Layer
7 (IMCL).

8

9 The modular architecture is implemented as a class that
10 defines a module, from which specific classes,
11 corresponding to modules with specific functionality, can
12 be derived. A module 112 is shown with its components
13 depicted. The module contains a number of components for
14 implementing functions:

- 15 1. A docking method 114 that is called when the
16 module is introduced to the agent and that describes
17 the functionality provided by that module;
- 18 2. An undocking method 116 that is called when the
19 module is required to detach itself from the agent
20 and terminate;
- 21 3. An update method 118 that is called when the
22 module is to be updated with a new version; and
- 23 4. A set of message handling method 120 for between-
24 agent communication.

25

26 The docking method of a particular module provides the
27 IMCL with a list of labels. The extensible set of these
28 labels is consistent system-wide, functioning as the
29 definition of the system's programming interface and is
30 maintained centrally by the service provider. A
31 particular module implements methods that match some
32 (typically small) subset of all labels. On docking, a
33 module informs the IMCL which labels it provides

1 functionality for. As each module docks, the IMCL builds
2 a table 122 of labels against which are listed the names
3 of the methods that provide the functionality and the
4 names of the modules implementing those methods.

5
6 In some cases, the labels implemented by a particular
7 newly docking module may already be implemented by
8 methods extant in the agent. So for example functionality
9 implemented by method x in module m1 may mean that when
10 m1 docks in an agent, it needs to register the label f
11 with the IMCL. But some other method, y, in some other
12 module, m2, that is already docked, may also implement
13 functionality f. Thus the IMCL would list functionality
14 f as being implemented by m2.y.

15
16 In such situations, the IMCL adds the new label and
17 associated method to its table, but also includes a
18 priority ranking, applied to both the new and old methods
19 that implement the functionality. This priority is
20 determined on the basis of some algorithm. Recency is a
21 good example, whereby newer methods are always given a
22 higher priority, but there are many such algorithms.

23
24 In addition, a module's docking method calls functions
25 provided by the core's module management. These calls
26 register information about the module with the core,
27 including its name, provenance and labelled
28 functionality.

29
30 A user interacts with the electronic world for a host of
31 reasons in a wide variety of domains: entertainment, e-
32 commerce, professional, and so on. The present invention
33 provides a means of bringing together all of these tasks

1 and domains, and providing a single point of contact for
2 the user, and allowing the sharing of user data between
3 these different application domains. This contact is the
4 user's agent, both in the computer-science sense (where
5 agent oriented programming has particular restrictions,
6 techniques and approaches, and places particular demands
7 on software), and also in the intuitive sense of
8 providing services of advocacy and representation. A
9 user's agent is their permanent representative in the
10 electronic world. Ideally, each user has exactly one
11 agent, and a user's agent represents exactly one user (at
12 the very least, such a relationship exists in a given
13 context). The overall picture is as in Figure 2.

14
15 With reference to Figure 2, an overview of agentative
16 representation in a multiservice environment is shown.
17 The user 202 connects to their agent 206 at any time via
18 any device (2G phones, multimedia mobile handsets,
19 internet-capable hardware, etc.) in ways that are well
20 known. The user agents 204 which represent users in the
21 virtual world are shown. One user has a single agent 206
22 representing him or her in all their interactions in the
23 virtual world. The service agents 208 provide specific
24 services to any agents that request them, or that the
25 service agents themselves decide to service. Information
26 exchange between user and service agents can be initiated
27 from either end. Some service agents 210 encapsulate
28 existing legacy services (e.g., databases, Web Services
29 and proprietary data handling systems). Broker agents
30 212 can mediate between a user and service agents. The
31 user agents service agents and broker agents may be
32 provided as a trusted service by a telecommunications
33 operator.

1
2 An agent is a software entity with particular
3 characteristics. We refer here to software processes that
4 are:

5 persistent (in that they continue to exist for an
6 extended real time period, adapting to a single user over
7 that time);

8 proactive (in that they include not only reactive
9 behaviour, but also independently determined behaviour);

10 communicative (in that they communicate with other
11 agents); and

12 autonomous (in that they typically cannot be directly
13 modified by outside agencies, but must instead be altered
14 through communication).

15
16 The user can communicate with his agent across
17 heterogeneous networks from a variety of devices,
18 including mobile handsets and internet clients. In
19 addition, however, the framework of the present invention
20 supports the transparent filtering of information
21 according to the device to which it is being sent. Thus
22 the components within an agent that initiate
23 communication with a user need not have any
24 representation of the device type a user is employing.
25 The content of the message is instead dynamically
26 tailored to the user's device (e.g. summary text to an
27 SMS-enabled mobile device, still pictures to a MMS-
28 enabled mobile device, streaming video to broadband
29 internet client platform, etc.).

30
31 The core is responsible for tailoring information to the
32 device that is known to currently be available to the
33 user. Thus, tailoring happens independently of the

1 module calls, so that individual modules do not need to
2 maintain device-specific information.

3
4 This filtering is achieved through a module-independent
5 communication object that is filled in by individual
6 modules when they need to communicate with the user.

7 This object has subparts for different forms of media
8 (text, picture, video, audio, etc.,). A module fills in
9 as many of these subparts as it is able. The core then
10 mediates the sending of that message to the user, by:

11 (i) identifying which device the user is currently
12 employing (using a combination of historical usage
13 patterns, presence information, and most recent-
14 communication data);

15 (ii) mapping the device to a set of media types (so,
16 e.g., an old phone can handle text, a newer device,
17 pictures);

18 further limiting the media types on the basis of user
19 preferences, and what has been made available by the
20 module; and

21 initiating the delivery of the appropriate media from the
22 user communication object constructed by the module.

23
24 In order to provide representation for a user, an agent
25 must implement a range of functionality. This
26 functionality is gathered together into the core module.
27 Modules can safely make the assumption that the core is
28 available for them to make calls upon.

29
30 The core contains a range of specific methods that
31 implement particular components of functionality. These
32 methods can be grouped together into functional groups.
33 Thus the core can be subdivided into discrete areas of

1 functionality. Any module can make a call on any of the
2 methods in any of the areas of the core's functionality
3 via the IMCL. The core provides methods that provide
4 functionality corresponding to a fixed set of labels
5 concerned with generic agent activity. This functionality
6 includes:

- 7 1. Belief management (including lookup and update)
- 8 2. User profile management (including lookup and
9 update)
- 10 3. Agent-User communication
- 11 4. Module Management
- 12 5. Basic generic reasoning tools
- 13 6. Between-Agent Module-Module communication (BAMM)
14 (send and receive)

15
16 The agent as a whole is a unitary autonomous software
17 entity, and as such maintains a single, coherent set of
18 tokens representing information about the world.
19 This belief database is changed through the action of
20 methods in the core. These methods implement core labels
21 for belief update. Any module (including the core itself)
22 can make calls as described herein on these labels
23 through the IMCL.

24
25 Similarly, the belief database can be queried by any
26 method through a call to a label mapped through the IMCL
27 to core functionality. Thus a module can perform a lookup
28 on the currently held beliefs by calling this label.

29
30 The user profile is a subset of the belief database, and
31 includes information specific to the user across a range
32 of domains. Again, the core implements labels
33 corresponding to update and query to the user profile.

1

2 There is the potential for the core to update the user
3 profile dynamically in response to user actions - that
4 is, the agent could adapt to and learn the user's
5 preferences as a result of repeated interaction.

6

7 User data (e.g., address; credit card details; age) and
8 user preferences (e.g., policy on releasing credit card
9 details; preference for aisle or window seat on planes;
10 preferred DVD supplier) are stored in a local, private,
11 secure database. Both user data and user preferences are
12 extracted in three ways. First, through an explicit
13 online interface that requests input on date of birth, or
14 supports update to reflect change of address. Second, if
15 the agent recognises information that it needs from the
16 user, it can ask for it directly (e.g. asking a yes/no
17 question by SMS). Third, as the user interacts with
18 services manually, the agent can intercept information
19 either explicitly or implicitly. If the user answers a
20 particular question from a particular online service, the
21 agent may either store that answer for future use, or ask
22 the user explicitly if such storage is appropriate or
23 useful. When acting autonomously, the agent provides
24 information that external service requires (and no more),
25 less anything that the user has placed a restriction on.
26 Thus, for example, when interacting with an online
27 newspaper, the newspaper provider may request user
28 registration, but not demand it. In this case, the agent
29 would provide no user information. Alternatively, when
30 interacting with a book e-tailer, the e-tailer may
31 require personal details including credit card data. If
32 the user has instructed his or her agent not to give out
33 credit card details without confirming it first, the

1 agent would halt interaction with that site until user
2 confirmation was sought and agreed.

3
4 These components could be represented by the steps:

- 5 1. Agent has goal of interacting with a service
- 6 2. Select required information from the user model
7 (UM) (accesses the UM)
- 8 3. Check that the user model permits all this
9 information to be freely given (accesses the UM)

10 If so,

- 11 4. Information given to the service

12 Otherwise

- 13 5. Process the restriction (either by terminating,
14 or by asking the user, or by performing some
15 other action)

16
17 The core also includes a subsystem responsible for
18 passing messages to, and receiving messages from the
19 user. The user may connect to his or her agent through a
20 number of different channels: using a web browser on a
21 PC, using a rich media mobile device (a Java phone, for
22 example), using a high capacity mobile device (such as
23 one that uses GPRS), or using an older, limited media
24 device (say that can only handle voice and SMS traffic).
25 The core implements labels that handle communication to
26 and from such devices quite transparently: the calling
27 module does not need to specify the different
28 communication types at all.

29
30 The means by which one agent communicates with another is
31 implemented in the core. Rather than supporting only
32 agent-to-agent messages, the architecture is instead
33 built around the idea that it is individual modules

1 within agents that communicate with one another (this is
2 "between agent module-module" or BAMB communication).
3 Thus a module with expertise in buying in a particular e-
4 commerce institution will communicate with a module in
5 another agent that has expertise in selling in that same
6 e-commerce institution. The fact that those agents also
7 happen to have modules with expertise in a range of other
8 diverse applications has no impact upon the conversation
9 between buyer and seller in this domain. It is thus
10 modules that structure conversations. The individual
11 utterances (or, more accurately, utterance types) that a
12 module uses to construct a given conversation are common
13 across the entire architecture. The sending and receiving
14 of these individual utterances is co-ordinated by the
15 core.

16

17 In this way, a module in an agent can conduct
18 conversations tailored to the domain in which the module
19 has competence. Though the conversation structure is
20 tailored, the implementation of primitive sending and
21 receiving is located in the core. This means that there
22 needs to be only one language definition - the language
23 that agents use for all communication. (If BAMB
24 communication was implemented solely in modules, those
25 modules would, by definition, use their own idiosyncratic
26 languages, and therefore the number of languages would be
27 proportional to the square of the number of module
28 types.) As language design and verification is a labour
29 intensive task, reducing the task by separating primitive
30 semantics from conversation definition, and rendering the
31 former once only in the core, saves a great deal of
32 effort.

33

1 The IMCL provides a small number of function calls, the
2 most important of which is the call which effects Within-
3 Agent Module-Module (WAMM) communication. When one module
4 wants to call a method in another module (including a
5 method provided by the core) it calls the IMCL's WAMM
6 communication method, passing it a label. The IMCL then
7 resolves that label by referring to its table of labels.

8
9 This means that one module need not know which other
10 module implements the functionality of a given label.
11 Indeed, a module can be implemented in such a way that it
12 can attempt a call on some labelled functionality, but
13 exhibits robustness in the event that no module is
14 present that implements that functionality. (Consider,
15 for example, module x that is, amongst other things,
16 responsible for performing some exponentiation
17 calculation. Module x has two ways of performing the
18 calculation - doing it itself, slowly and laboriously
19 using repeated addition, or by asking a specialised
20 module y that can do exponentiation quickly and
21 efficiently. The problem is that x has no way of knowing
22 whether or not y is installed. Thus x makes a call to the
23 IMCL requesting exponentiation on a particular data set.
24 If y is installed, the IMCL will pass the request to the
25 appropriate method within y. If y is not installed, the
26 IMCL will inform x that no module implements
27 exponentiation and x can then follow the more laborious
28 route of performing the calculation itself). The process
29 by which a label is resolved is summarised in Figure 3.

30
31 With reference to Figure 3, a module makes a call to
32 label L 310. The IMCL looks up L in a label table 312.
33 If L is not present 314, the IMCL returns "not found"

1 316. If L is present, and L does have multiple
2 resolutions 318, then the IMCL selects the highest
3 priority resolution 320. Next the IMCL calls the method
4 described in the resolution 322. Finally, when the
5 method returns a value 324, the IMCL passes the return
6 value back to the caller.

7
8 A practical advantage of the approach is that it removes
9 compile time dependencies: a module developer can design,
10 implement and test a module which makes calls to another
11 module that they do not have, or do not have access to,
12 or, indeed, that has not been developed at all. This
13 simplifies many of the problems of software engineering
14 in the large, and of multi-site collaborative development
15 work.

16
17 With reference to Figure 4, a flowchart 700 of the module
18 docking process is shown. First, either a user specifies
19 702 a module, the core receives a message 704 specifying
20 a module to download, or the core determines 706 that a
21 module should be downloaded. The core reasons and decides
22 708 whether or not to download the module. If a download
23 is determined to be completed, the core calls 710 the
24 IMCL with the *getmodule* label and URI (Universal Resource
25 Identifier) specifying the location of the Java Archive
26 file (having the .jar file extension). If not, the
27 docking process ends 730. The IMCL then resolves 712 the
28 *getmodule* label to the *ModuleManagement.ModuleGet* method
29 in the core and the IMCL calls 714 the core's
30 *ModuleManagement.ModuleGet* method with the URI.
31 *ModuleManagement* downloads 716 the Java Archive file from
32 the URI. The .jar manifest file 718 specifies the Module
33 class and the Module class 720 includes the *DockMe*

1 method. With this information, the ModuleManagement
2 instantiates 722 the Module class and calls 724 the
3 DockMe method. The DockMe method may then register 726
4 labels with the IMCL and initialise 728 the BMM handler
5 before ending the process 730.

6
7 The undocking method of a particular module informs the
8 IMCL and the core module management that the
9 functionality provided by the module is terminating, and
10 then kills the module's thread.

11
12 One of the advantages of decoupling of functionality in
13 separate modules is in allowing modules to be added to an
14 agent on-the fly. Such modules need not have
15 functionality that is known before the agent is
16 instantiated, as each module describes its own behaviour
17 and functionality on docking. The same approach can be
18 used to update existing modules with new versions.

19
20 Another, practical, advantage of the approach is that it
21 removes compile time dependencies: a module developer can
22 design, implement and test a module which makes calls to
23 another module that they do not have, or do not have
24 access to, or, indeed, that has not been developed at
25 all. This simplifies many of the problems of software
26 engineering in the large, and of multi-site collaborative
27 development work.

28
29 The core is responsible for the handling of new modules
30 and module updates. It arranges for outgoing requests to
31 be sent for new modules (or for updated versions of
32 existing modules), handles incoming requests for the
33 agent to take on new modules, and handles the process by

1 which a module is added. So, for example, if the user
2 decides to sign up for a new, free service, the core may
3 send a message to the coordinating agent for that
4 service, requesting the module. After some brief
5 negotiation, the coordinating agent may send back a
6 request to install the new module. The core module
7 management accepts and downloads the software. After
8 updating the internal module database, it starts the new
9 module, which then docks with the IMCL, as described
10 above.

11
12 The process of updating a module and the handover between
13 the older and newer versions is illustrated in Figure 5.
14 With reference to Figure 5, a flowchart 800 of the module
15 updating process is shown. First, either a user specifies
16 802 a module, the core receives a message 804 specifying
17 a module to update, or the core determines 806 that a
18 module should be updated. The core reasons and decides
19 808 whether or not to update the module. If an update is
20 determined to be completed, the core calls 810 the IMCL
21 with the *getmodule* label and URI specifying the location
22 of the Java Archive file. If not, the updating process
23 ends 836. The IMCL then resolves 812 *updatemodule* label
24 to the *ModuleManagement.ModuleUpdate* method in the core
25 and the IMCL calls 814 the core's
26 *ModuleManagement.ModuleGet* method with the module name
27 and the URI. *ModuleManagement* calls 816 the module's
28 *UpdateMe* method and the *UpdateMe* method returns the
29 module state information, which is then stored 818 by
30 *ModuleManagement*. *ModuleManagement* downloads 820 the Java
31 Archive file from the URI. The .jar manifest file 822
32 specifies the Module class and the Module class 824
33 includes the *DockMe* method. With this information, the

1 ModuleManagement instantiates 826 the Module class and
2 calls 828 the DockMe method with the stored state
3 information. The DockMe method may then register 830
4 labels with the IMCL and initialise 832 the BMM handler
5 before the Module Management calls 834 the UnDockMe
6 method on the old module and unloads the class then ends
7 the process 836.

8
9 The core's module management also handles more complex
10 situations, in which a new version of an existing module
11 is installed. The handover between the modules requires
12 careful timing to ensure service to the rest of the agent
13 are not disrupted. When a new version of a module is to
14 be docked, the core module management calls the old
15 version's update method, and, at an appropriate moment
16 (at a break between discrete tasks that the module is
17 working on) the module returns a representation of its
18 current state.

19
20 Then, as part of the new module's docking method, it
21 accepts the state of the previous version's data, so that
22 it can pick up from where the previous module left off.

23
24 During the handover between versions of a module, the
25 IMCL queues calls to the modules' labels, awaiting the
26 docking of the new version.

27
28 Some of the functionality described herein is collected
29 together in the core. Rather than hardwire a distinction
30 between core and other modules, the core is instead
31 implemented in just the same way as any other module.
32 This means that calls on core functionality can exploit

1 the robustness of the label-based, IMCL-mediated method
2 calling.

3
4 It also means that core itself can be updated and docked
5 through just the same process as any other module (so the
6 activity of the module management component in the old
7 core is interrupted and then taken up by the activity of
8 the module management component in the new core). This
9 means that new core functionality can be deployed
10 dynamically, without other modules needing to know ahead
11 of time all that might be included.

12
13 Agentative representation in mobile services represent a
14 domain in which persistency is crucial, and quality of
15 service requirements are extremely high (and in some
16 cases, are at safety critical levels). The present
17 invention provides the capability to decouple modules in
18 such a way that both data exchange and method calls can
19 be executed without prior knowledge of anticipated
20 functionality.

21
22 In addition, agents can be enabled to be dynamically
23 equipped with behaviours that had no definition at the
24 time those agents were initiated. Furthermore the agents
25 are provided with means for: (i) agent mediation of the
26 update (that is agents communicate between themselves to
27 arrange for update to functionality); (iii) dynamic
28 update of core functionality in exactly the same way as
29 any other module (because core should be treated in the
30 same way as any other module); and (iv) the ability to
31 replace existing functionality in a principled manner, as
32 well as add to it.

33

1 Further modifications and improvements may be added
2 without departing from the scope of the invention herein
3 described.
4
5